#7

PATENTTI- JA REKISTERIHALLITUS
NATIONAL BOARD OF PATENTS AND REGISTRATION

Helsinki 13.12.2001

E T U O I K E U S T O D I S T U S
P R I O R I T Y D O C U M E N T

Hakija Applicant	**Republica Jyväskylä Oy** **Jyväskylä**
Patenttihakemus nro Patent application no	**20010136**
Tekemispäivä Filing date	**23.01.2001**
Kansainvälinen luokka International class	**G06F**

Keksinnön nimitys
Title of invention

"Menetelmä ja laitteisto tiedon uudelleenryhmittelemiseksi"

Täten todistetaan, että oheiset asiakirjat ovat tarkkoja jäljennöksiä patentti- ja rekisterihallitukselle alkuaan annetuista selityksestä, patenttivaatimuksista, tiivistelmästä ja piirustuksista.

This is to certify that the annexed documents are true copies of the description, claims, abstract and drawings originally filed with the Finnish Patent Office.

Marketta Tehikoski
Apulaistarkastaja

Maksu 300 mk (50 € 1.1.2002 lähtien)
Fee 300 FIM (50 EUR from 1 January 2002)

Maksu perustuu kauppa- ja teollisuusministeriön antamaan asetukseen 1027/2001 Patentti- ja rekisterihallituksen maksullisista suoritteista muutoksineen.

The fee is based on the Decree with amendments of the Ministry of Trade and Industry No. 1027/2001 concerning the chargeable services of the National Board of Patents and Registration of Finland.

Osoite: Arkadiankatu 6 A
P.O.Box 1160
FIN-00101 Helsinki, FINLAND
Puhelin: 09 6939 500
Telephone: + 358 9 6939 500
Telefax: 09 6939 5328
Telefax: + 358 9 6939 5328

Menetelmä ja laitteisto tiedon uudelleenryhmittelemiseksi – Metod och apparatur för att regruppera data

Keksintö koskee sellaisten sääntöjen muodostamismenetelmää, jonka avulla tietoa voidaan ryhmitellä uudelleen. Lisäksi keksintö koskee järjestelyä tällaisen menetelmän toteuttamiseksi.

Tietoa on saatavilla valtavasti, koska sähköistä dataa kokoavia, tehokkaita laitteita on paljon ja tallennuskapasiteetti on kasvanut. Sähköiseen dataan pääsee helposti käsiksi, mutta oleellisen tiedon löytäminen onkin jo vaikeampaa. On olemassa tekniikoita, joiden avulla voidaan ennakoida ja arvioida datan sisältöä. Yleensä halutaan löytää voimassaolevaa, uutta, käyttökelpoista, ymmärrettävää tietoa. Ongelmia ilmenee esimerkiksi, jos data on epätäydellistä tai epäyhtenäistä. Data voi myös olla väliaikaista tai muuttuvaa, sitä voi olla suunnaton määrä käsiteltäväksi tai se voi olla muutakin kuin tekstimuotoista. Suuresta datamäärästä pitäisi löytää tiettyyn yritykseen tai tiettyyn ongelmaan liittyvä spesifinen tieto.

Eri lähteistä löytyvää tietoa on tarpeen myös yhdistää ja vertailla. Jotta epäyhtenäistä tietoa voidaan analysoida, on tieto saatettava yhtenäiseen muotoon. Lisäksi yhtenäistämistä edellyttää yritysten sisäinen ja yritysten kesken tapahtuva tiedonkulku ja -vaihto. Nykyään persoonallisuus on tärkeä erottautumiskeino markkinoilla. Persoonalliset ratkaisut kuitenkin eroavat niin sanotuista standardiratkaisuista ja vaikeuttavat siten tiedon jatkokäsittelyä. Dokumentit tulisikin esittää niin joustavasti, että ne soveltuvat ja ovat yhdistettävissä mahdollisimman moniin muihin sovelluksiin. Yleensä yksittäinen yksilöllinen ratkaisu vaatii tehokkaat laitteet hoitamaan yhteyksiä. Yhteyksistä huolehtivien laitteiden ja ohjelmistojen on oltava helppokäyttöisiä ja niiden on kasvettava yrityksen mukana. Tällaisen laitteistokokonaisuuden ylläpito ja ohjelmointi kaikki standardit ja vaatimukset huomioon ottaen on hyvin raskasta.

On olemassa ratkaisuja, jotka yhtenäistävät erinäisiä tiedostoformaatteja samaan, käsiteltävään muotoon. Esimerkiksi XWrap Elite (an eXtensible Wrapper Generation System Elite Version) on ohjelmistosovellus, joka pyrkii tunnistamaan toistuvat tietorakenteet HTML-dokumenteista ja luomaan niille kullekin muunnossäännöt, joilla niiden sisältö saadaan muunnettua XML-kielelle. Sovelluksen toiminta on kuitenkin sidottu HTML–rakenteeseen. XWrap Elite-ohjelmistoa on ennen tämän hakemuksen prioriteettipäivää kuvattu yksityiskohtaisemmin osoitteessa http://www.cc.gatech.edu/projects/disl/XWRAPElite/. Vastaavanlainen HTML-

rakenteeseen pohjautuva ratkaisu on myös W4F – World Wide Web Wrapper Factory, jota on ennen tämän hakemuksen prioriteettipäivää kuvattu tarkemmin osoitteessa http://www.tropea-inc.com/technology/W4F/. Lisäksi Xeroxin Euroopan tutkimuskeskuksessa on keväällä 2000 julkistettu tiedon muunnosskriptin generointimenetelmä, Wrapper Generation via Grammar Induction, joka on myös sidottu HTML-formaattiin. Tämä menetelmä on ennen tämän hakemuksen prioriteettipäivää julkaistu osoitteissa http://turing.wins.uva.nl/~ragetli/documents/chidlovskii.ps ja http://turing.wins.uva.nl/~ragetli/ecml2000/ecml00a/. Kaikki nämä mainitut, olemassa olevat ratkaisut perustuvat HTML-formaattiin, joten niiden käyttöalue on melko rajoitettu. Aiheesta on olemassa myös patentti US6151608, jossa kuvataan tiedon muuntamista tietokantataulukkoon koodia kirjoittamatta.

Keksinnön tavoitteena on tuottaa menetelmä ja järjestely, joilla ohjelmointitaidotonkin käyttäjä voi muodostaa mistä hyvänsä sähköisestä tietovirrasta valitsemilleen data-alueille irrotussäännöstön. Säännöstön avulla etsitään ja irrotetaan data-alueita lähdedatasta.

Tavoite toteutetaan asteittain oppivan koodingenerointisovelluksen avulla.

Keksinnön mukaisesti käyttäjä saa koodia kirjoittamatta valitunlaiset data-alueet erotettua lähdedatasta jatkokäsittelyä varten.

Koodingenerointisovellus voi käyttää tiedon esittämiseen ja valintojen tekemiseen datavirrasta mitä tahansa selainta. Selaimella tarkoitetaan tässä sellaista tiedon tarkasteluvälinettä, jonka esittämään tietoon voidaan kohdistaa valintoja. Jos kyseessä on esimerkiksi HTML- tai tekstiselain, voidaan yhtenäinen jatkuva alue valita perinteisesti "hiirellä maalaamalla". Käyttäjä toimii selaimelle ominaisessa ympäristössä, eikä tarvitse välttämättä lainkaan koodingenerointisovelluksen tuottamaa tiedon irrottamiseen käytettävää lähdekoodia. Käyttäjällä on olemassa dataa, jotka hän jatkokäsittelyä varten haluaa eritellä tietyin osin ja/tai yhtenäistää. Nyt käyttäjän tarvitsee vain osoittaa koodingenerointiohjelmalle haluamastaan lähdedatasta kaksi tai useampia esimerkkejä. Esimerkit voidaan näyttää myös eri lähdedatoista.

Käyttäjä antaa ohjelmalle esimerkkitapaukset, joiden perusteella ohjelma osaa irrottaa lähdedatasta tietyt rakenneosaset. Lähdedatasta käyttäjä osoittaa koodingenerointiohjelmalle tietyt osaset, joiden mallin mukaan ohjelma muodostaa säännöstön. Säännöstön perusteella koodingenerointiohjelma etsii ja irrottaa tietoa lähdedatasta. Käyttäjä voi tämän jälkeen esimerkiksi yhtenäistää epäyhtenäistä dataa samaan formaattiin. Käyttäjän antamien esimerkkien tulisi olla sisällöltään mahdollisimman

erilaisia tapauksia samantyyppisen, tunnistettavan tietorakenteen osasta. Ne voivat olla vaikkapa taulukon rivejä, luettelon alakohtia tai lomakkeita.

Koko sähköisestä tietotulvasta käyttäjä siis valitsee halutut datalähteet tai halutut osat datalähteistä ja osoittaa ne syötteenä koodingenerointiohjelmalle. Tämän tuloksena käyttäjä saa lähteisiin perustuvan tiedon järjestettynä sekä tuotetun irrotussäännöstön. Tulosdata voidaan esittää myös halutussa formaatissa, jonka käyttäjä voi joko määrittää itse tai valita valmiista esimerkkimäärityksistä. Lisäksi ohjelma esittää valitut piirteet helposti tarkasteltaviksi.

Seuraavassa selostetaan keksintöä yksityiskohtaisemmin viitaten esimerkkinä esitettyyn edulliseen suoritusmuotoon ja oheisiin kuviin, joissa

kuva 1 esittää keksinnön perusperiaatetta,
kuva 2 esittää erään edullisen sovellusmuodon mukaista muunnosskriptin luomista ja
kuva 3 esittää erään edullisen sovellusmuodon mukaista tokenisoitujen esimerkkien käsittelyä.

Hajanaisia, eri muodoissa olevia dokumentteja ja tiedostoja on usein tarpeen yhtenäistää, jotta niitä voidaan jatkokäsitellä. Yhtenäistäminen on tärkeää yritysten välisiä asioita käsiteltäessä, koska on käytännössä mahdotonta vaatia, että kaikki yhteistyö- tai asiakassuhteessa toimivat yritykset yhtenäistäisivät tietojärjestelmänsä ja ohjelmistonsa. Yhtenäistettävyys tietojen käsittelyssä helpottaa paitsi yritysten sisäistä tiedonkäsittelyä, erityisesti julkaisu- ja yhteistyöverkostojen toimintaa. On oleellista, että sähköisestä informaatiotulvasta voidaan valita itselle tärkeät alueet ja että näitä osioita voidaan tarkastella, vertailla ja käsitellä edelleen yhtenäisenä kokonaisuutena.

Kuvassa 1 on esitetty keksintö pääpiirteissään. Koodingenerointisovellus on datan erotteluun, järjestämiseen ja yhtenäistämiseen sovitettu työkalu. Koodingenerointisovellus käsittää kolme osaa: koodingenerointikomponentin 102, muodostettavat irrotussäännöt 103 sekä irrotuskomponentin 104. Lähdemateriaali 101 voi olla mitä hyvänsä tietoteknistä materiaalia, kuten esimerkiksi tiedostoja, dokumentteja tai jatkuvaa tietovirtaa. Ainoa edellytys lähdeformaatille on, että välittömästi jokaisen automaattisesti irrotettavaksi halutun tietokentän edellä tai jälkeen on oltava vähintään yhden käsiteltävän datan perusyksikön eli tokenin mittainen kenttäerotin, joka toistuu samanlaisena kaikissa tietueissa juuri ennen kyseistä tietokenttää tai välittömästi sen jälkeen. Itse irrotettava tietue voi kuitenkin sisältää erilaisia kenttäerotti-

mia. Käyttäjä valitsee koko lähdemateriaalin tai tiettyjä lähdemateriaalin osia, data-alueita. Näiden valittujen data-alueiden tietorakenteen tulee olla litteä eli tietorakenteessa ei saa esiintyä eri hierarkiatasoja. Tällainen litteä tietorakenne on esimerkiksi taulukolla tai listalla. Jos käyttäjä haluaa käsitellä vain tiettyä osaa tietorakenneyksiköstä, vaikkapa taulukkoa, hänen tulee antaa koodingenerointisovelluksen koodingenerointikomponentille 102 esimerkkinä vähintään kaksi sellaista taulukon riviä, jotka eroavat toisistaan mahdollisimman paljon. Mitä enemmän esimerkit poikkeavat toisistaan, sitä vähemmän esimerkkejä tarvitaan toivotunlaisen irrotustuloksen aikaansaamiseksi. Valittujen esimerkkirivien mahdollisimman monessa sarakkeessa tulee siis olla eri tietoa.

Koodingenerointisovelluksen irrotussäännöt tuottava komponentti 103 muodostaa esimerkkien perusteella säännöstön, jonka perusteella irrotuskomponentti 104 etsii ja irrottaa halutunlaisia data-alueita lähdemateriaalista 101. Varsinainen irrotuskomponentti voi olla mikä hyvänsä asiaan soveltuva komponentti. Irrotuksen tuloksena saadut data-alueet voidaan jatkokäsitellä halutulla tavalla, esimerkiksi tallentaa. Mikäli ne tallennetaan tiettyyn yhteen kohteeseen, käyttäjä saa dokumentin 105, joka voi olla käytännössä millainen hyvänsä riippuen käyttäjän valinnoista ja alkuperäisistä lähdedatoista. Alkuperäinen dokumentti säilyy muuttumattomana. Tuotettu tulos sisältää käyttäjän syöttämien esimerkkien mukaiset data-alueet, esimerkiksi kaikki taulukon rivit, käyttäjän määrittämässä muodossa ja järjestyksessä. Käyttäjä voi määrittää itse esimerkiksi kohdeformaatin muutaman parametrin avulla tai valita valmiista, olemassa olevista formaateista. Jos käyttäjä valitsee valmiin formaatin, formaatti määritetään esimääriteltyjen parametrien avulla. Saatavaa kohdedataa varten käyttäjä voi määritellä myös kenttien järjestyksen ja/tai jättää haluamansa tietokentät kokonaan pois käytöstä.

Sitä osaa lähdedatasta, joka valitaan muunnettavaksi, kutsutaan data-alueeksi. Tyypillisesti data-alue on jokin alue, jonka sisällä toistuu vain yksi tietorakenne. Data-alue voi olla esimerkiksi taulukko, joka siis sisältää taulukossa toistuvat rivit. Data-alue voi myös olla esimerkiksi kappale tekstiä, otsikko tai luettelo. Data-alueet voivat olla keskenään täysin erilaisia, mutta kunkin data-alueen sisällä toistuu yksi säännöllinen tietorakenne. Tämä toistuva tietorakenne on helposti osoitettavissa koodingenerointisovellukselle ja edelleen koodingenerointisovelluksen löydettävissä ja erotettavissa lähdedatasta. Taulukossa tällainen toistuva ominaisuus olisi esimerkiksi tietosisällön järjestäytyminen riveihin ja/tai sarakkeisiin. Tekstikappaleita puolestaan ryhmittelee yleensä niiden alussa ja lopussa olevat tyhjät tilat ennen seuraavan osion alkua. Otsikko on varustettu tietyillä tunnistettavilla ominaisuuksilla,

kuten esimerkiksi lihavoinnilla ja/tai tekstin sijoittelulla. Luettelo sisältää yleisesti jonkin tunnistesymbolin, esimerkiksi viivan tai laatikon ennen varsinaista asiasisältöä ja rivinvaihdon sen jälkeen. Oleellista siis on, että on olemassa jokin tietyille osioille yhteinen tietorakenne, jonka koodingenerointisovellus löytää käyttäjän antamien esimerkkien avulla.

Data-alueita voidaan jakaa osiin myös sisällön perusteella. Näitä vaihtelevan tai kiinteän pituisia yksiköitä kutsutaan tokeneiksi. Menetelmää, jonka avulla merkkijonoja jaetaan osiin sisällön perusteella, kutsutaan tokenisoinniksi. Yksi tokeni sisältää tyyppimerkinnän eli nimen sekä itse datasisällön. Oletuksena on, että yksi tokeni eli käsittelyn yksikkö on yksittäinen merkkijärjestelmän symboli, kuten kirjain, numero tai välimerkki. Usein osoittautuu kuitenkin kannattavammaksi määritellä tokenit siten, että yksi tokeni sisältää tietyn datassa toistuvan merkkijonon. Tokenityypit ovat siis eräänlaisia yleistyksiä. Esimerkiksi "numero"-tyyppisiä tokeneja ovat kaikki numerot, jos tokenityypin säännöllinen lauseke on "[0-9]" tai jos "sana"-tyyppisen tokenin säännöllinen lauseke on "[A-Za-z]+", tokeni voi sisältää yhden tai useamman peräkkäisen merkin väliltä A – Z tai a – z. On myös mahdollista yksilöidä, että esimerkiksi vain ja ainoastaan merkkijono "B2B" on "test2"-tyyppinen tokeni. Tokenisoidut merkkijonot ovat helpommin hallittavissa kuin yksittäiset symbolit.

Tokenisointi helpottaa ja nopeuttaa datan käsittelyä. Käytetyn tokenisoinnin ansiosta nyt kyseessä oleva sovellus ei ole sidottu mihinkään tiettyyn formaattiin, vaan riittää, että irrotettavaksi valitun tietokentän edellä tai sen jälkeen on vähintään yhden tokenin mittainen kenttäerotin. Tokenisointi siis monipuolistaa ja nopeuttaa koodingenerointisovelluksen käyttöä, sillä vähäinenkin pohjatieto käsiteltävästä formaatista tokenisointimäärityksekseksi kirjoitettuna mahdollistaa parempiin irroitussääntöihin pääsemisen pienemmillä esimerkkimäärillä. Vastaavasti virheellisen tokenisoinnin käyttö voi johtaa siihen, ettei algoritmi pysty tuottamaan ollenkaan toimivia irrotussääntöjä. Tokenisoimalla data ja suorittamalla tämän jälkeen kaikki käsittelyoperaatiot tokenisoituun dataan voidaan varmistaa koodingenerointisovelluksen oikeanlainen toiminta. Tokenisointia hyväksikäyttäen saavutetaan myös kulloiseenkin tilanteeseen sopiva tarkkuus lähdemateriaalin tarkastelulle. Tässä keksinnössä tokenisointi on apuväline irrotussääntöjen generoinnissa, mutta sääntöjä tulkitseva tiedon irrotuskomponentti ei enää tiedä tokenisoinnista mitään, eikä käytä sitä mihinkään.

Käyttäjä osoittaa koodingenerointisovellukselle esimerkit. Esimerkkien osoittamisella tarkoitetaan tässä, että käyttäjä osoittaa esimerkit olemassa olevasta datasta ja/tai

tuottaa esimerkit itse ja/tai muokkaa olemassa olevia esimerkkitapauksia korostaakseen niiden yhteneviä osia.

Käyttäjän koodingenerointisovellukselle antamien esimerkkien tulisi olla mahdollisimman erisisältöiset ei-yhteneviltä osiltaan, jotta koodingenerointisovellus yksiselitteisesti löytäisi esimerkkitapausten yhtenäiset osat, joita se käyttää luodessaan datanirrotussäännöstöä. Koska esimerkkien on oltava mahdollisimman erilaiset sisällöltään, tokenisoinnin avulla mahdollistetaan tietynlaisten datalähteiden käyttö. Esimerkiksi HTML-tiedosto voidaan jakaa tageihin ja niiden välisiin tekstipaloihin. Jos HTML-formaattia käsiteltäisiin ilman tokenisointia, eli oletusarvoisesti yksi symboli muodostaisi yhden tokenin, esimerkkien osoittaminen koodingenerointisovellukselle olisi työlästä, jollei mahdotonta. Käyttäjältä kuluisi kohtuuttomasti aikaa sopivien esimerkkien löytämiseen, koska näiden sisältöjen tulisi olla symbolitasolla täysin erilaiset. Esimerkkitiedon yksilöimiseksi tarvittaisiin siis hurjan paljon esimerkkejä tai sitten niiden laatua tulisi voida oleellisesti parantaa. Tämä laadun parannus tehdään nimenomaan tokenisoimalla: kun HTML-formaatin mukainen data on tokenisoitu tageiksi ja niiden väliin jääviksi merkkijonoiksi, yhdenkin yksittäisen symbolin ero merkkijonojen sisällä riittää ilmaisemaan koodingenerointisovellukselle, että alue halutaan irrottaa ja säilyttää yhtenäisenä.

Tässä tekstissä yksittäisestä merkkijärjestelmän merkistä käytetään nimitystä "symboli". Useammasta peräkkäisestä symbolista käytetään kuitenkin nimitystä "merkkijono". Mainitut irrotettavien data-alueiden alku- ja loppumerkit eivät ole yksittäisiä symboleja, vaan merkkijonoja tai säännöllisiä lausekkeita. Jokerimerkkikään ei välttämättä ole vain yksi merkki (*), vaan sitä voi vastata jokin merkkijono (esimerkiksi .*) järjestelmästä, ohjelmointikielestä ja sovelluksesta riippuen.

Koodingenerointisovelluksen käyttäjä voi halutessaan itse määritellä tokenisointisäännöt. Yleensähän tokenisointisäännöt valitaan valmiista asetuksista, joista esimerkkinä HTML-kieleen sopivasta tagien ja niiden väliin jäävien merkkijonojen erittely. Käyttäjä voi myös muokata valmiita tokenisointisääntöjä.

Esimerkkinä esimerkkitapausten erilaisuudesta voidaan tarkastella tilannetta, jossa esimerkkitapauksia on neljä ja kukin niistä koostuu kolmesta peräkkäisestä kentästä. Se, että kussakin esimerkkitapauksessa on kolme kenttää, tarkoittaa, että esimerkkitapauksilla on samanlainen rakenne. Sisällön erilaisuus tarkoittaa, että jos tarkastellaan yhdessä kaikkien esimerkkitapausten ensimmäisiä (tai toisia, tai kolmansia) kenttiä, ne eivät ole kaikissa esimerkkitapauksissa samat. On kuitenkin sallittua, että vaikkapa ensimmäisen esimerkkitapauksen ensimmäinen kenttä on sama kuin toisen

ja kolmannen esimerkkitapauksen ensimmäinen kenttä, kunhan neljännen esimerkkitapauksen ensimmäinen kenttä on sisällöltään erilainen.

Koodingenerointisovellus tallentaa käyttäjän syöttämät esimerkit, joten esimerkkejä voidaan lisätä yksi kerrallaan valiten niitä eri lähteistä. Tyypillisesti myös irrotussäännöt tallennetaan, jolloin samoja, tallennettuja sääntöjä voidaan käyttää myöhemminkin tarvitsematta generoida niitä uudelleen.

Käyttäjän lähdedatasta valitsema esimerkki on merkkijono. Tällaisen irrotettavan merkkijonon edellä tai sen jälkeen on oltava vähintään yhden tokenin mittainen kenttäerotin, joka toistuu samanlaisena kaikissa tietueissa. Tietueen sisällä kenttäerottimien ei kuitenkaan tarvitse olla samoja. Tietorakenteen perusteella irrotettavissa olevat kentät paikallistetaan ja yksilöidään, eli kentille annetaan järjestysnumero. Tietokenttien rajat määräytyvät kulloinkin käytettävän tokenisoinnin mukaisesti. Käyttäjän antamien esimerkkien perusteella koodingenerointisovellus muodostaa editointiskriptin (edit script). Editointiskriptissä luetellaan ne muutokset, joilla ensimmäisen syötetyn esimerkkitapauksen sisältö saadaan samaksi kuin toisen esimerkkitapauksen sisältö. Editointiskriptissä toistuva perusyksikkö on muutostieto Tämä muutostieto sisältää indeksiosoittimet molempiin merkkijonoihin sekä korvattavan ja korvaavan tiedon kokonaisuudessaan. Kohdistamalla määrätyllä tavalla editointiskriptien muutokset pisimpään syötettyyn esimerkkilausekkeeseen, jota kutsutaan säännölliseksi lausekkeeksi, muodostetaan tuloksena saadun säännöllisen lausekkeen avulla irrotusskripti, joka sitten etsii lähdedatasta merkkijonoa syötetietojen perusteella. Edelleen irrotusskriptin mukaiset tietyt, valitut data-alueet irrotetaan lähdedatasta jatkokäsittelyä varten.

Vaikka esimerkit koodingenerointisovellukselle näytettäisiin tiedostoista, varsinainen tiedon irrotus generoitujen sääntöjen avulla voidaan tehdä tiedostojen lisäksi myös esimerkiksi jatkuvasta, päättymättömästä tai tuntemattoman pituisesta tietovirrasta. Irrotettavan data-alueen pituus tiedetään siis vasta siinä vaiheessa, kun tietovirrasta tai erillisessä kontrollivirrasta löydetään jokin ennaltamäärätty lopetusmerkki.

Kun käyttäjä on valinnut esimerkkitapaukset, hän voi testata yksittäisen data-alueen irrotussääntöjen toimintaa esikatselu-tilassa koodingenerointisovelluksen selaimessa. Jos käyttäjä havaitsee, että ennen hänen esimerkiksi tarkoittamaansa data-aluetta on jotain ylimääräistä dataa, käyttäjä voi määritellä data-alueen alku- ja loppumerkit tai yrittää täsmentää olemassa olevia. Käyttäjän tulee valita ennen toivottua irrotettavaa data-aluetta edeltävä merkkijono siten, että se olisi mahdollisimman yksiselit-

teinen ja täsmällinen kertomaan sovellukselle, että juuri tällaisen tietyn merkkijonon jälkeen tulevaa dataa lähdemateriaalista halutaan irrottaa. Näin käyttäjä voi esikatselu-tilassa tarkistaa, että valittujen esimerkkitapausten tyyppistä tietoa irrotetaan oikeilta alueilta. Vastaavasti voidaan varmistaa sovelluksen toiminta data-alueen loppumerkin osalta.

Esimerkit voidaan myös syöttää täysin erillisinä vaikkapa suoraan näppäimistöltä. Kertaalleen osoitettuja esimerkkejä on mahdollista muuttaa: jos käyttäjä huomaa yhden kokeilun jälkeen, ettei kahden esimerkin välillä ollut riittävää sisällöllistä eroa, hän voi lisätä eroja syöttämällä ainakin toiseen esimerkkiin joitain sellaisia symboleja, joita toisessa ei ole.

Koodingenerointisovelluksen selaimen ominaisuuksista riippuu, näkyvätkö data-alueen alku- ja loppumerkkejä kuvaavat merkkijonot käyttäjälle suoraan lähdedatasta ja/vai korostetaanko niiden välinen alue selkeästi havaittavaksi. Tarkoitushan on, että lähdemateriaalin alusta lähtien ensimmäinen löydetty alkua tarkoittava merkkijono on irrotettavaksi haluttua data-aluetta edeltävä kohta. Alkumerkistä eteenpäin lähdemateriaalista etsitään loppumerkkiä, johon irrotettava data-alue päättyy. Jos data-alueen alku- tai loppumerkki puuttuu eli on tyhjä, koodingenerointisovellus tulkitsee irrotettavan data-alueen aluksi lähdemateriaalin alun ja vastaavasti irrotettavan data-alueen lopuksi lähdemateriaalin lopun. Data-alue-määritys on siis aina olemassa, eikä se ole millään tavoin riippuvainen lähdemateriaalin formaatista, rakenteesta tai sisällöstä. Käyttäjä voi myös muokata tiedon irrotussäännöstön ohjelmakielistä koodia, jos esimerkiksi sopivia esimerkkejä ei löydy riittävästi.

Jos käyttäjä haluaa tuloksena tiedon yhtenäisessä muodossa, käyttäjä voi määritellä kohdeformaatin muutamalla parametrilla tai valita jonkin valmiin, kuten esimerkiksi XHTML-taulukon, pohjaksi. Halutessaan käyttäjä voi siis määrittää, missä muodossa ja millä lisämerkinnöillä varustettuina irrotetut tietueet kirjoitetaan kohteeseen. Jos tuloksena saatavaa kohdedataa halutaan käsitellä esimerkiksi XML-selaimella, voidaan valmiiksi määrittää tiettyä lähdemateriaalista irrotettua osaa ympäröivät tagit, jotta tuleva kohdedata on heti havainnollisemmassa ja helpommin käsiteltävässä muodossa. Seuraavassa on esimerkkinä XML-formaatin mukaisia tageja ja niiden välissä se lähdemateriaalin osa, jonka tagit ympäröivät:

<datastream>koko kohdedata</datastream>
<dataarea>data-alue</dataarea>
<record>tietue</record>
<field>kenttä</field>
<address>kenttä</address>
<price>kenttä</price>.

Data-alueella tarkoitetaan siis esimerkiksi tauluja tai taulukoita, joista kukin sisältää yhden tai useampia tietueita eli taulun/taulukon rivejä, joista kukin sisältää yhden tai useampia kenttiä eli taulun/taulukon rivien soluja/sarakkeita.

Keksintö ei sinänsä edellytä, että irrotetut tietueet tallennetaan johonkin tiettyyn yhteen kohteeseen yhtenäisessä formaatissa. Kun tieto on irrotettu lähteestä, sitä voidaan jatkokäsitellä monin tavoin. Esimerkkejä mahdollisesta jatkokäsittelystä ovat irrotetun tiedon antaminen jollekin toiselle ohjelmistokomponentille tai laitteelle, irrotetun tiedon lähettäminen eteenpäin esimerkiksi verkkoyhteyttä pitkin ja irrotetun tiedon osittainen tai täydellinen hävittäminen. Irrotetut data-alueet ja/tai niiden osat (tietueet, kentät) voidaan irrotuksen jälkeen järjestää uudelleen erilaisiin järjestyksiin joko ennen niiden jatkokäsittelyä tai osana jatkokäsittelyä.

Tarkastellaan yksityiskohtaisemmin koodingenerointisovelluksen toimintaa ja muunnosskriptin tuottamista yhden data-alueen osalta kuvan 2 avulla. Koodingenerointisovellukselle osoitetaan syötteenä lista data-aluemäärityksiä, joista jokainen sisältää vähintään kaksi esimerkkiä ja merkkijonon tai säännöllisen lausekkeen valitun data-alueen alkamis- ja päättymiskohdan tunnistamiseksi. Tietorakenne-esimerkkien sisälle ei tarvitse erikseen laittaa mitään aloitus- tai lopetusmerkkejä. Aluksi kohdassa 201 tokenisoidaan kaikki käyttäjän syöttämällä esimerkkilistalla olevat esimerkit. Kuvan algoritmi ei tarvitse data-alueiden alku- ja loppumäärityksiä esimerkkien käsittelyyn, vaan se yksinkertaisesti lisää data-alueen rajaamiseksi tarvittavat, valmiina saamansa tiedot irrotusskriptiin.

Seuraavaksi 202 esimerkit järjestetään tokenimääräisesti laskevaan pituusjärjestykseen. Kohdassa 203 listan ensimmäinen ja siis pisin esimerkki kopioidaan ja se määritetään säännölliseksi lausekkeeksi R. Tämän jälkeen käsitellään listan seuraavaa esimerkkilausetta 204. Seuraavaa, toiseksi pisintä lausetta merkitään esimerkkilausekkeella E. Kohdassa 205 verrataan säännöllistä lauseketta R esimerkkilausekkeeseen E käyttämällä sinänsä tunnettua Diff-vertailualgoritmia, jonka tarkempi kuvaus ennen tämän hakemuksen prioriteettipäivää löytyy osoitteesta http://www.cs.arizona.edu/people/gene/PAPERS/diff.ps. Tämä vertailualgoritmi

palauttaa lyhimmän mahdollisen editointiskriptin D eli säännöllisen lausekkeen R ja esimerkkilausekkeen E välisen muutostiedon. Jos toteutetaan editointiskriptin D sisältämät muutokset säännölliselle lausekkeelle R, saadaan esimerkkilause E.

Keksintö ei sinänsä edellytä nimenomaan tunnetun Diff-algoritmin käyttöä, vaan sen tilalla voidaan käyttää mitä tahansa sellaista vertailevaa algoritmia, joka palauttaa editointiskriptiä vastaavat ohjeet siitä, miten tietystä kohteesta saadaan lyhimmällä mahdollisella tavalla muuntamalla tietty toinen kohde. Jos on olemassa vähintään kaksi lyhintä mahdollista tapaa, keksinnön kannalta ei ole väliä sillä, mikä näistä tavoista valitaan. Lyhin mahdollinen tapa tarkoittaa, että ns. editointietäisyys eli tarvittavien muutosten lukumäärä muunnettaessa lähdettä kohteeksi on mahdollisimman pieni.

Seuraavaksi 206 luetaan Diff-vertailualgoritmin palauttamasta editointiskriptistä D muutosoperaatiot yksi kerrallaan. Editointiskriptiä D ja säännöllisen lausekkeen R sisältämiä tokeneja vertaillaan keskenään 207. Säännöllistä lauseketta R muutetaan siten, että ne tokenit, joihin D:stä luettu muutostieto viittaa, merkitään jokerimerkillä, jolloin jäljelle jäävät alkuperäisinä vain R:n muuttumattomat tokenit. Seuraavaksi tutkitaan, onko kaikki D:n muutosoperaatiot jo läpikäyty. Jos muutosoperaatioita on vielä lukematta, palataan kohtaan 206. Jälleen tokenit, joihin muutostieto kohdistuu, merkitään säännöllisessä lausekkeessa R jollain tietyllä jokerimerkillä (*-wildcard).

Kohdassa 209 tarkastetaan, onko esimerkkilistalla vielä lisää esimerkkilausekkeita. Kaikki listan esimerkit käydään yksi kerrallaan läpi eli jos lisäesimerkkejä löytyy, jatketaan kohdasta 204. Mikäli kaikki esimerkkilausekkeet on jo käsitelty, edetään kohtaan 210 ja poistetaan tuotetusta säännöllisestä lausekkeesta peräkkäiset jokerimerkit siten, että vain ensimmäinen niistä jää jäljelle. Seuraavaksi säännöllinen lauseke R muunnetaan osaksi muunnosskriptiä 211. Alun perin (ennen tätä vaihetta) muunnosskripti on tyhjä. Tuotettava muunnosskripti etsii lähdemateriaalista merkkijonoja syöttötietojen perusteella. Kun säännöllisessä lausekkeessa on jokerimerkki, lähdemateriaalista etsitään se merkkijono, joka säännöllisessä lausekkeessa on jokerimerkin jälkeen. Jokerimerkki tavallaan korvaa pienimmän mahdollisen määrän mitä hyvänsä lähdemateriaalin merkkejä siihen merkkijonoon asti, joka säännöllisessä lausekkeessa esiintyy jokerimerkin jälkeen. Tällaista jokerimerkin tulkintaa nimitetään usein reluctant-tulkinnaksi.

Muunnosskriptin eteen lisätään vielä kohdan 212 mukaisesti koodi, jolla yksi datalue kokonaisuutena irrotetaan lähdedatasta. Tämän lähdedatan sisältöä koodingenerointisovellus tässä esimerkissä käsittelee ja muuntaa halutut osat siitä kohdefor-

maatin mukaisiksi. Tämän jälkeen muunnosskriptiin voidaan vielä lisätä tämän esimerkin mukaisesti käyttäjän mahdollisesti antamat tai ennaltamääritetyt parametrit, jotka määrittävät kohdeformaatin. Koodingenerointisovellus siis tuottaa muunnosskriptin, joka irrottaa kaiken data-alueilla esiintyvän annettuihin tietorakenne-esimerkkeihin täsmäävän tiedon jatkokäsittelyä varten, joka tässä esimerkkitapauksessa on konvertointi yhtenäiseen kohdeformaattiin.

Eräs vaihtoehto muunnosskriptin automaattiselle muodostamiselle kohtien 211 ja 212 mukaisesti on jatkaa kohdasta 210 siten, että muodostettu säännöllinen lauseke R otetaan talteen ja käsitellään muulla tavoin kuin liittämällä se osaksi muunnosskriptiä. Se voidaan esimerkiksi käyttää "manuaalisesti" uudelleen jonkin toisen ohjelmatuotteen osana tai sitä voidaan soveltaa pelkkänä irrotusskriptinä tekemättä konvertointia kohdeformaattiin.

Tarkastellaan vielä kuvan 3 avulla säännöllisen lausekkeen muodostamista. Tässä esimerkkitapauksessa käyttäjä on osoittanut koodingenerointisovellukselle kolme esimerkkilauseketta, jotka on tokenisoitu. Kaksi pisintä esimerkkilauseketta koostuu 8 tokenista ja lyhin esimerkkilauseke 6 tokenista. Säännölliseksi lausekkeeksi R 301 merkitään pisin esimerkkilauseke eli tässä tapauksessa kumpi hyvänsä pidemmistä 8 tokenin esimerkkilausekkeista. Jäljelle jäänyt pisimmistä esimerkkilausekkeista merkitään E:ksi 302. Lausekkeet R ja E osoitetaan parametreina tunnetulle Diff-vertailualgoritmille, joka palauttaa editointiskriptin D 303. Nyt editointiskriptin muutostiedot on esitetty siten, että kunkin rivin ensimmäiset numerot viittaavat säännöllisen lausekkeen R 301 korvattaviin tokeneihin ja seuraavat numerot esimerkkilausekkeen E 302 korvaaviin tokeneihin. Lisäksi editointiskripti D 303 sisältää niiden tokenien sisällön, joissa muutoksia tapahtuu. Kuvan 3 editointiskriptin D 303 mukaisesti säännöllisen lausekkeen R 301 tokeni R[2] muutetaan vastaamaan esimerkkilausekkeen E 302 tokeneja E[2] ja E[3]. Säännöllisen lausekkeen R 301 tokenit R[5] ja R[6] on korvattava esimerkkilausekkeen E 302 tokenilla E[6]. Kaikki muuttuvat tokenit on kuvassa ilmaistu katkoviivanuolella ja samana pysyvät, toisiaan vastaavat tokenit on kuvattu yhtenäisellä nuolella.

Seuraavaksi muodostetaan uusi säännöllinen lauseke R' 304 siten, että muuttumattomat tokenit säilyvät säännöllisessä lausekkeessa R 301 ennallaan ja kaikki esimerkkilausekkeesta E 302 eroavat, editointiskriptin mukaisen muutostiedon mukaan muuttuvat, katkoviivanuolella merkityt tokenit korvataan jokerimerkillä, joka tässä esimerkissä on *. Uudeksi esimerkkilausekkeeksi E' 305 merkitään seuraava käyttäjän syöttämä esimerkki, minkä jälkeen tuotetaan R' 304 ja E' 305 välinen muutostieto D' 306 Diff-vertailualgoritmin avulla. Diff-vertailualgoritmin palauttamasta edi-

tointiskriptistä D' 306 luetut muutokset on merkitty katkoviivanuolilla ja R':n 304 ja E':n 305 muuttumattomat tokenit on yhdistetty yhtenäisellä nuolella. Lisäksi editointiskriptin toinen muutostieto koskee tokenia R[4], jolla ei ole vastinetta esimerkkilausekkeessa E' 305. Tällöin tokeni R[4] korvataan tyhjällä ja seuraavaa säännöllistä lauseketta muodostettaessa sitä käsitellään kuten muitakin korvattavia tokeneja. Muodostetussa säännöllisessä lausekkeessa R'' 307 ovat jäljellä muuttumattomat tokenit R[1], R[3], R[7] ja R[8]. Kaikki muuttuneet tokenit on korvattu valitulla jokerimerkillä *. Tämän jälkeen säännöllisestä lausekkeesta R'' 307 poistetaan peräkkäiset jokerimerkit (ei esitetty kuvassa) siten, että vain ensimmäiset niistä, jäävät jäljelle. Esimerkki lausekkeessa R'' 307 on nyt neljä kenttäerotintokenia, joiden väliin muodostuu kaksi eri kenttää. Ensimmäinen varsinaista tietosisältöä sisältävä kenttä on tokenien R[1] ja R[3] välissä ja toinen kenttä sijaitsee tokenien R[3] ja R[7] välissä. Kenttäerottimet R[7] ja R[8] esiintyvät peräkkäin ja muodostavat tämän säännöllisen lausekkeen loppumerkin. Tämän jälkeen säännöllinen lauseke R'' 307 on valmis ja voidaan esimerkiksi lisätä osaksi irrotusskriptiä.

Mikäli jossakin esimerkkilausekkeessa ei ole mitään rakennetta, edellä selostettu toiminta johtaa siihen, että säännöllinen lauseke R'' on lopulta pelkkä yksi jokerimerkki. Tällöin tuloksena syntyvä muunnoskripti irrottaa koko sen data-alueen, johon irrotussäännöstöä yritetään soveltaa. Irrotuksen tuloksena kaikki irrotettu tieto sijaitsee yhden irrotetun data-alueen yhden tietueen yhdessä kentässä.

Koodingenerontisovelluksen koodingenerointikomponentti (102) tuottaa koodikielisen irrotusskriptin, jonka pohjalta varsinainen irrotuskomponentti suorittaa valikoivaa tietojen irrotusta sisään tulevasta datasta. Edullisesti irrotusskripti tuotetaan jollain tunnetulla koodikielellä sovelluksesta riippuen. Edullisesti koodingenerointisovellus toimii kaikissa Java 1.2.2 tai sitä uudemmissa versioissa. Tulosteena saadaan irrotetut tiedot koodikielisen skriptin määrittämässä muodossa ja järjestyksessä. Tässä esimerkinomaisesti mainittu DEL (Data Extraction Language) -kieli on tehty nimenomaan ja ainoastaan tiedon irrotusta varten, mutta se ei millään tavoin rajoita keksintöä käyttämään vain tätä tiettyä kieltä, vaan edullisesti voidaan käyttää esimerkiksi Perl-, Python-, REBOL- tai OmniMark-suorittimia. Kieli toimii formaatin konvertoijan tavoin eli se ei irrota esimerkiksi lauseen subjektia, vaan toimii symbolitasolla. Säännöllisiin lausekkeisiin pohjautuvien irrotussääntöjen muokkaamiseen on oma editorinsa, joka riippuu kulloinkin tiedon irrotukseen käytettävästä komponentista. Irrotettava tietokenttä on merkkijono, joka voidaan skriptiä muokkaamalla asettaa siirrettäväksi haluttuun kohtaan koodingenerointisovelluksen irrotuskomponentin (104) tuottamaa tulosdataa.

Koodingenerointisovellus voi käyttää tiedon näyttämiseen ja tiedosta valintojen tekemiseen lähes mitä hyvänsä selainta. Selaimelta edellytetään, että sen datan lataamiseen (tiedoston/tietovirran avaaminen/sulkeminen), esittämiseen ja valitsemiseen liittyvä toiminnallisuus on jonkinlaisen ohjelmointirajapinnan kautta ohjattavissa ja säädeltävissä, ja että kaikki tiedot käyttäjän tekemistä valinnoista saadaan heti tapahtumahetkellä koodingenerointisovelluksen käytettäväksi. Käytetystä selaimesta riippuu, miten ja millaisia osia jatkuvasta tietovirrasta esitetään käyttäjälle. On myös mahdollista käyttää useampia selaimia yhtäaikaisesti siten, että esimerkiksi kulloinkin käsiteltävä lähdedata sijaitsee omassa selaimessaan, valitut esimerkit ovat talletettuna toisessa selaimessa ja tiedon irrotuksen tulosten esikatselu käyttää kolmatta selainta.

Tiedon irrotussääntöjen generointi sekä tiedon irrotus ja konversio haluttuun kohdeformaattiin haluttaessa voidaan toteuttaa palveluna verkon yli. Tällöin joko koodingenerointikomponentti, irrotuskomponentti tai molemmat voivat sijaita palvelimella muun sovelluksen toimiessa käyttäjän työasemalla.

Sovelluksen suorituksen tuloksena saadaan siis halutun sisältöinen tietopaketti. Käyttäjän ei tarvitse osata ohjelmoida tai muuntaa sovellusta, vaan hän saa esimerkkiensä mukaisen tuloksen mistä hyvänsä haluamastaan lähdedatasta. Saatuja tietoja voidaan edelleen jalostaa, tilastoida ja julkaista yhtäaikaisesti eri muodossa käyttäen ulkoisia sovellusohjelmia. Sovellus on tehokas ja monipuolinen. Se on helposti siirrettävissä ja ylläpidettävissä nykyisissä laitteistoissa, koska edullisen suoritusmuodon mukaisesti käytetty Java-pohjainen DEL-suoritin on helposti integroitavissa eri systeemeihin. Edullisesti sovelluksen käyttöliittymä toimii esimerkiksi Windows 2000/NT-ympäristössä, johon on asennettu Microsoftin Internet Explorer-selaimen versio 5 tai jokin uudempi ratkaisu. Lisäksi koodingenerointisovellus on mukautuva ja helposti uudelleen käytettävissä ihan uudenlaisenkin lähdemateriaalin kanssa.

Patenttivaatimukset

1. Menetelmä tietyssä alkuperäisessä datassa sijaitsevan tiedon käsittelemiseksi, **tunnettu** siitä, että menetelmä sisältää vaiheet, joissa
- osoitetaan vähintään kaksi esimerkkitapausta (301, 302),
- osoitettujen esimerkkitapausten (301, 302) perusteella muodostetaan säännöstö (103) halutunlaisen tiedon irrottamiseksi, ja
- muodostetun säännöstön mukaiset data-alueet alkuperäisestä tietueesta irrotetaan (104).

2. Patenttivaatimuksen 1 mukainen menetelmä, **tunnettu** siitä, että irrotetut data-alueet muunnetaan formaatiltaan yhtenäisiksi (105).

3. Patenttivaatimuksen 1 mukainen menetelmä, **tunnettu** siitä, että osoitetuilla vähintään kahdella esimerkkitapauksella on kullakin rakenne ja sisältö, jolloin rakenne on esimerkkitapausten välillä samanlainen mutta sisältö on erilainen.

4. Patenttivaatimuksen 1 mukainen menetelmä, **tunnettu** siitä, että esimerkkitapaukset osoitetaan alkuperäisestä datasta (101).

5. Patenttivaatimuksen 1 mukainen menetelmä, **tunnettu** siitä, että esimerkkitapausten (301, 302) perusteella muodostettu säännöstö (103) tallennetaan myöhempää käyttöä varten.

6. Patenttivaatimuksen 1 mukainen menetelmä, **tunnettu** siitä, että osoitetut esimerkkitapaukset tokenisoidaan (201) ennen niiden varsinaista käsittelyä korvaamalla tietyt esimerkkitapausten osiot niitä vastaavilla tietorakenteilla, jotka sisältävät tunnisteen, kuten tyyppimerkinnän tai nimen, sekä kyseisen osion datasisällön.

7. Patenttivaatimuksen 6 mukainen menetelmä, **tunnettu** siitä, että osoitetuissa vähintään kahdessa esimerkkitapauksessa (301, 302) on vähintään yksi samanlainen osio, jota esimerkkitapausten käsittelyssä vastaa tietty tokeni.

8. Patenttivaatimuksen 6 mukainen menetelmä, **tunnettu** siitä, että säännöstön muodostamiseksi
- pisin osoitetuista tokenisoiduista esimerkeistä merkitään säännölliseksi lausekkeeksi R (203, 301),
- seuraavaksi pisintä osoitetuista tokenisoiduista esimerkeistä merkitään esimerkkilausekkeeksi E (204, 302) ja

– verrataan säännöllistä lauseketta R (203) ja sen hetkistä esimerkkilausetta E (204).

9. Patenttivaatimuksen 8 mukainen menetelmä, **tunnettu** siitä, että säännöllistä lauseketta R ja sen hetkistä esimerkkilauseketta E vertaillaan tietyn vertailualgoritmin avulla (205), joka palauttaa editointiskriptin D (206, 303, 306).

10. Patenttivaatimuksen 9 mukainen menetelmä, **tunnettu** siitä, että säännöllistä lauseketta R ja sen hetkistä esimerkkilauseketta E vertaillaan sellaisen vertailualgoritmin avulla (205), joka palauttaa lyhimmän mahdollisen editointiskriptin.

11. Patenttivaatimuksen 9 mukainen menetelmä, **tunnettu** siitä, että säännöstön muodostamiseksi säännöllistä lauseketta R muunnetaan editointiskriptin D (207) sisältämien muutostietojen pohjalta.

12. Patenttivaatimuksen 8 mukainen menetelmä, **tunnettu** siitä, että muodostettu säännöllinen lauseke muodostaa säännöstön (211).

13. Patenttivaatimuksen 1 mukainen menetelmä, **tunnettu** siitä, että muodostetun säännöstön avulla alkuperäisestä datasta irrotetaan esimerkkitapausten mukaiset osiot (104).

14. Laitteisto tietyssä alkuperäisessä datassa sijaitsevan tiedon käsittelemiseksi, **tunnettu** siitä, että laitteisto sisältää välineet

– vähintään kahden esimerkkitapauksen (301, 302) osoittamiseksi alkuperäisestä datasta (101),
– säännöstön muodostamiseksi osoitettujen esimerkkitapausten (301, 302) perusteella halutunlaisen tiedon irrottamiseksi, ja
– muodostetun säännöstön mukaisten data-alueiden irrottamiseksi (104) alkuperäisestä lähdemateriaalista.

15. Patenttivaatimuksen 14 mukainen laitteisto, **tunnettu** siitä, että siinä on välineet irrotettujen osioiden muuntamiseksi formaatiltaan yhtenäisiksi (105).

16. Patenttivaatimuksen 14 mukainen laitteisto, **tunnettu** siitä, että esimerkkitapausten osoittamista varten laitteistossa on osoittimet merkkijonoihin.

17. Patenttivaatimuksen 14 mukainen laitteisto, **tunnettu** siitä, että laitteisto sisältää välineet osoitettujen esimerkkien tokenisoimiseksi korvaamalla tietyt esimerkkitapausten osiot niitä vastaavilla tietorakenteilla, jotka sisältävät tyyppimerkinnän tai nimen sekä kyseisen osion datasisällön.

18. Patenttivaatimuksen 17 mukainen laitteisto, **tunnettu** siitä, että laitteisto sisältää välineet tokenisoidun datan käsittelemiseksi.

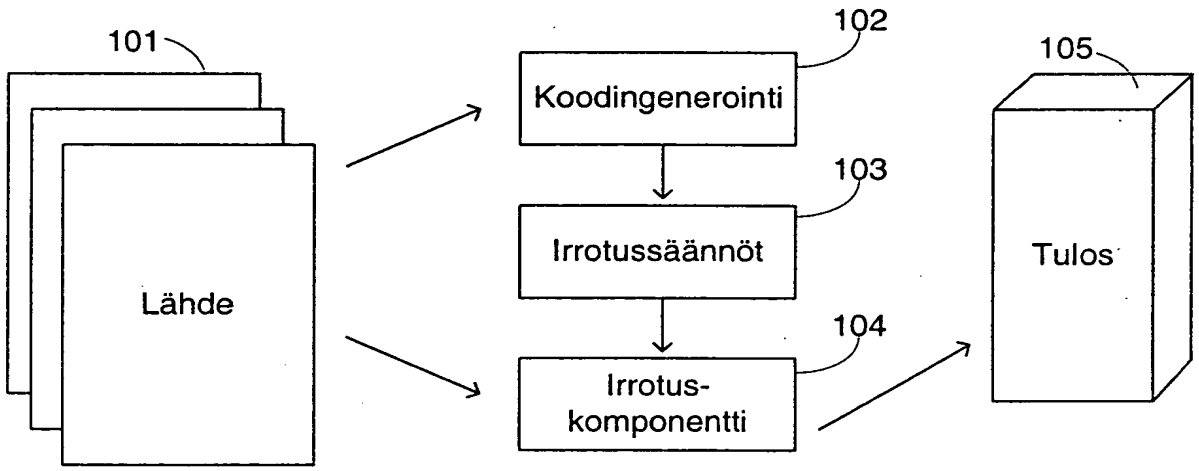
19. Patenttivaatimuksen 14 mukainen laitteisto, **tunnettu** siitä, että laitteisto sisältää välineet säännöstön muodostamiseksi muodostetun säännöllisen lausekkeen R mukaisesti.

20. Patenttivaatimuksen 19 mukainen laitteisto, **tunnettu** siitä, että välineet säännöstön muodostamiseksi sisältävät tätä tarkoitusta varten luodun ohjelmistokomponentin (102), joka on erillään sellaisesta ohjelmistokomponentista (104), joka on tarkoitettu data-alueiden irrottamiseen muodostettua säännöstöä käyttäen.

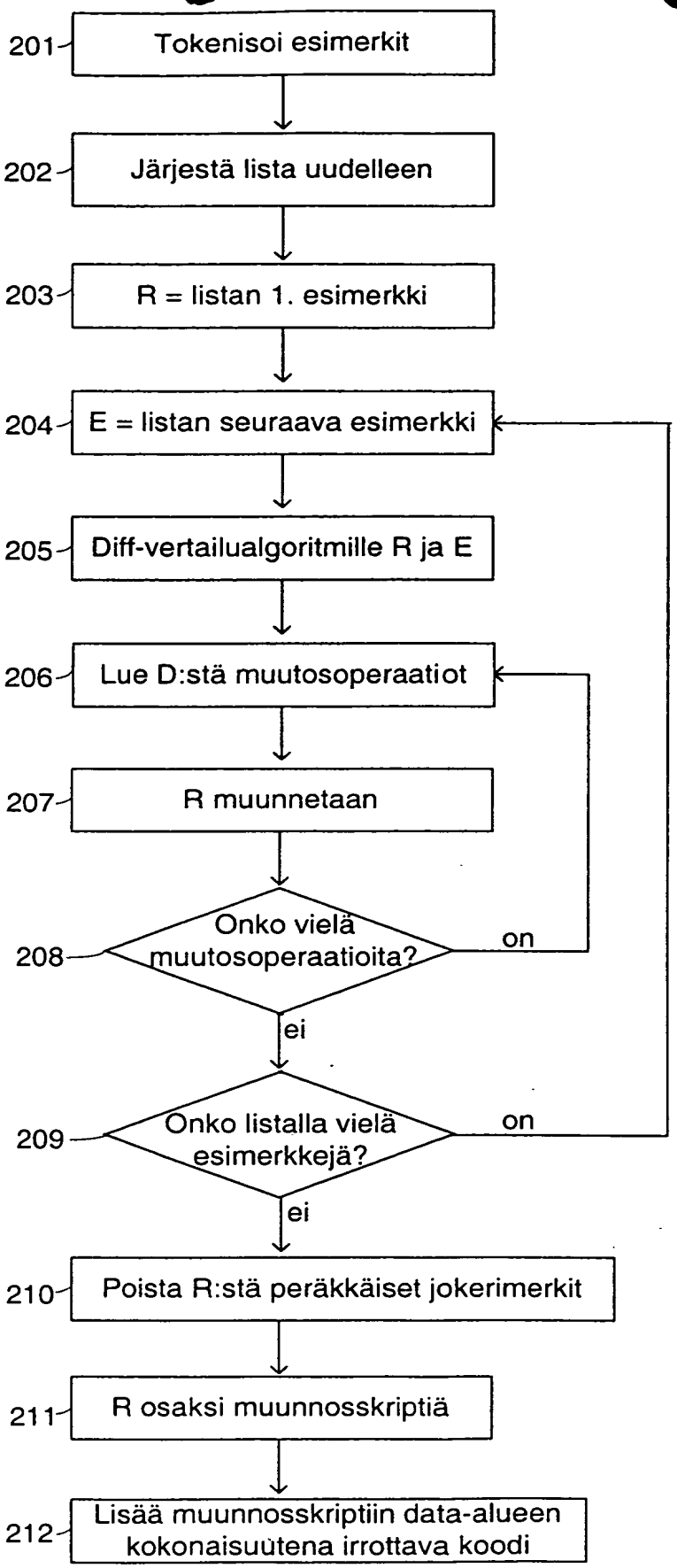
(57) Tiivistelmä

Keksintö koskee sellaisten sääntöjen muodostamismenetelmää, jonka avulla tietoa voidaan ryhmitellä uudelleen. Lisäksi keksintö koskee järjestelyä tällaisen menetelmän toteuttamiseksi. Keksinnön tavoitteena on tuottaa menetelmä ja järjestely, joilla ohjelmointitaidotonkin käyttäjä voi muodostaa lähdedatasta valitsemilleen data-alueille irrotussäännöt. Kyseessä olevassa menetelmässä alkuperäisessä datassa olevaa tietoa käsitellään siten, että alkuperäisestä lähdedatasta (101) käyttäjä valitsee vähintään kaksi esimerkkitapausta (301, 302). Osoitettujen esimerkkitapausten perusteella muodostetaan säännöstö (103), jonka mukaiset data-alueet alkuperäisestä lähdedatasta irrotetaan (104). Irrotettuja data-alueita (105) voidaan jatkokäsitellä halutulla tavalla.

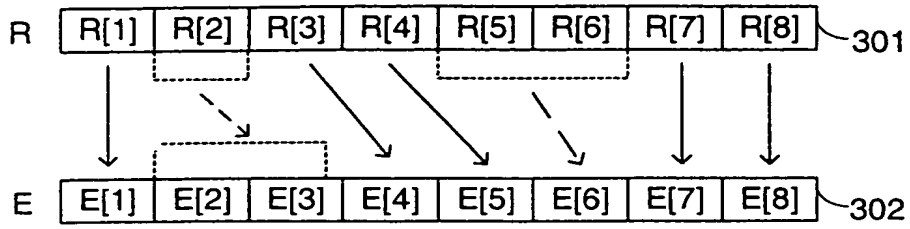
Kuva 1

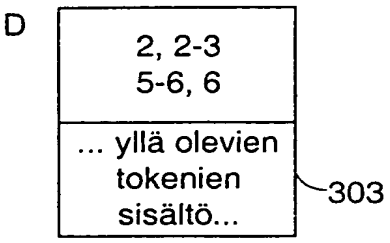

101
Lähde
102
Koodingenerointi
103
Irrotussäännöt
104
Irrotus-
komponentti
105
Tulos

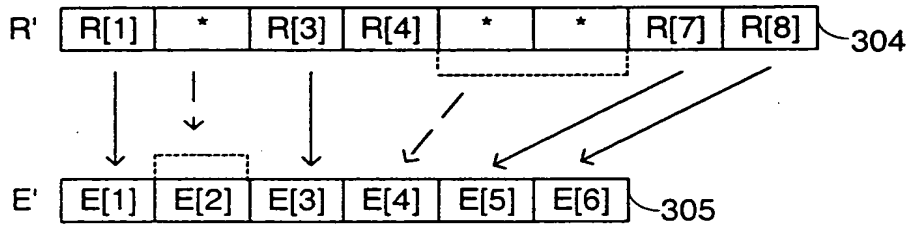
Kuva 1.

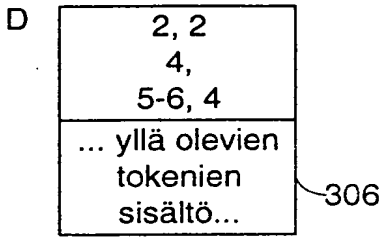

201
Tokenisoi esimerkit
202
Järjestä lista uudelleen
203
R = listan 1. esimerkki
204
E = listan seuraava esimerkki
205
Diff-vertailualgoritmille R ja E
206
Lue D:stä muutosoperaatiot
207
R muunnetaan
208
Onko vielä muutosoperaatioita?
on
ei
209
Onko listalla vielä esimerkkejä?
on
ei
210
Poista R:stä peräkkäiset jokerimerkit
211
R osaksi muunnosskriptiä
212
Lisää muunnosskriptiin data-alueen kokonaisuutena irrottava koodi

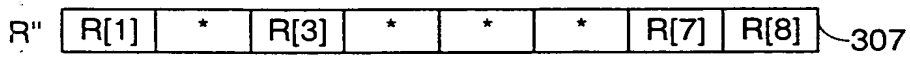
Kuva 2.


R
R[1]
R[2]
R[3]
R[4]
R[5]
R[6]
R[7]
R[8]
301
E
E[1]
E[2]
E[3]
E[4]
E[5]
E[6]
E[7]
E[8]
302


D
2, 2-3
5-6, 6
... yllä olevien tokenien sisältö...
303


R'
R[1]
*
R[3]
R[4]
*
*
R[7]
R[8]
304
E'
E[1]
E[2]
E[3]
E[4]
E[5]
E[6]
305


D
2, 2
4,
5-6, 4
... yllä olevien tokenien sisältö...
306


R"
R[1]
*
R[3]
*
*
*
R[7]
R[8]
307

Kuva 3.